

mm  
a

## METHOD AND APPARATUS FOR CONFIGURATION MANAGEMENT FOR A COMPUTING DEVICE

### BACKGROUND OF THE INVENTION

5

#### I. Field of the Invention

The method and apparatus for configuration management relates generally to the field of software management and more particularly to a method and apparatus for providing configuration management of software used for a general computing device.

#### II. Description of the Related Art

15 In modern computing systems, it is often important to effectively manage a software configuration that is embodied within a computing system. Such computing systems include not only traditional computers found in businesses and homes, but also a wide variety of electronics, including wireless telephones, laptop computers, personal digital assistants (PDAs), automotive electronics, and  
20 so on. Such computing systems sometimes employ methods to ensure software compatibility by checking a current software version with that of a newer version of software to be loaded. However, these methods do not attempt to prevent an upgrade of existing software, or the introduction of new software, under certain conditions.

25 One example of a need to manage and control the type or version of software to be loaded onto a computing device can be found in security applications for such computing devices. For example, in a governmental organization where each person is given a wireless communication device, the organization might be segregated into different security levels. If a person  
30 having a lower security level were to acquire a wireless communication device belonging to a person having a higher security level, that person might try to load software onto the wireless communication device so that he might be able to access information that would be otherwise unavailable to him. In another aspect of this example, it would be desirable to only load authenticated software into the  
35 wireless communication device once the wireless communication device has been given to a person having a high security level. Authenticated software refers to software that has been distributed by a "trusted" source, and that it has not been altered.

Thus, there is a need for a method and apparatus to perform configuration management and control for software used in a computing device.

## SUMMARY OF THE INVENTION

5

The problem of performing configuration management and control for software used in a computing device is solved, in one embodiment, by a method for software configuration management, comprising the steps of providing available software to an interface associated with a computing device, then determining whether or not resident software stored in an application storage area associated with the computing device has been authenticated. If the resident software has not been authenticated, the available software is loaded onto the computing device. If the resident software has been authenticated, the available software is loaded only if it has also been authenticated.

15

In another embodiment, the problem of performing configuration management and control for software used in a computing device is solved by an apparatus for software configuration management, comprising an interface for providing available software to the computing device and a storage device for storing resident software and data. The apparatus further comprises a processor for executing a set of computer instructions for determining whether or not the resident software is authenticated. The processor loads the available software if the resident software has not been authenticated. If the resident software has been authenticated, the available software is loaded only if it has also been authenticated.

25

## BRIEF DESCRIPTION OF THE DRAWINGS

The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings wherein:

30

FIG. 1 illustrates a computing device in functional block diagram format; and

35

FIG. 2 is a flow diagram illustrating one embodiment of the method for providing configuration management for a computing device.

## DESCRIPTION OF THE PREFERRED EMBODIMENT

A method and apparatus for configuration management for a computing device is presented. It should be understood that the disclosed method and apparatus may be applied to a variety of computing devices, including portable and desktop computers, mainframe computers, personal digital assistants (PDAs), wireline and wireless communication devices offering voice and/or data communication services, to name a few. In general, the disclosed method and apparatus may be applied to any electronic device requiring software to perform an intended task.

FIG. 1 illustrates computing device 100 in functional block diagram format. In one embodiment, computing device 100 comprises processor 102, interface 104, storage device 112 comprising application storage area 106, programmable storage area 108, and permanent storage area 110. It should be understood that other configurations are possible, and that the method and apparatus described herein could be implemented in any number of possible configurations.

As stated earlier, computing device 100 comprises any electronic device which requires software to perform its intended task. In one embodiment, computing device 100 comprises a wireless data device such as a wireless telephone having data capabilities to perform such tasks as manage schedules, access the Internet, send and receive email, and so on. The remaining discussion herein will describe the method and apparatus for providing configuration management for a computing device with respect to such a wireless communication device. It should be understood that the method and apparatus for providing configuration management for a computing device could also be applied to any computing device which uses software to perform its intended function.

Processor 102 comprises a digital processor for executing one or more sets of executable software instructions stored in storage device 112. In one embodiment, processor 102 comprises a digital microprocessor such as one from the 80x86 family of processors from Intel Corporation of Santa Clara, California. In another embodiment, processor 102 comprises a digital signal processor (DSP), such as the TMS320 series from Texas Instruments Incorporated of Dallas, Texas. In another embodiment, processor 102 comprises a custom ASIC. Other configurations known in the art may be used in the alternative.

Computing device 100 is initially provisioned with executable computer instructions to allow certain core activities to occur, for instance, just after computing device 100 is powered on. These executable computer instructions are herein referred to as a kernel. The kernel may perform a variety of self-tests by instructing executable computer instructions stored in either application storage

area 106, programmable storage area 108, or the kernel may execute its own series of self-tests. In addition, the kernel may initiate other software programs such as an operating system, such as Windows CE, used to host other software programs. The kernel also comprises executable computer instructions for providing configuration management of software as it is loaded into computing device 100.

Computing device 100 can generally run one or more software programs, either simultaneously or in sequence with each other. For example, an email program and an Internet browser could be executed simultaneously by processor 102. Other examples of software that may be operated by computing device 100 include word processing software, spreadsheet software, communication software, encryption software, etc.

In one embodiment, a user desiring to add software to computing device 100 provides the software to computing device 100 using interface 104. The software to be loaded is referred to herein as the available software. Interface 104 comprises an apparatus for allowing communication between computing device 100 and an external electronic device, such as a second computing device. Interface 104 comprises a disk drive, a parallel port, or a serial port, or any other electronic interface which allows software to be loaded onto computing device 100. In another embodiment, interface 104 comprises a wireless communication system for receiving software over-the-air.

Instructions are given to processor 102, either automatically by the presence of available software at interface 104, or by the user explicitly instructing processor 102 using an I/O device such as a keypad and display, to load the available software from interface 104 into storage device 106.

Processor 102, upon receiving instructions for loading the available software at interface 104, may temporarily accept the software and store it in programmable storage area 108 within storage device 112. Storage device 112 comprises one or more electronic memories for storing executable software programs and supporting data. More specifically, programmable storage device 108 comprises a random access memory (RAM), flash RAM, electrically erasable programmable read-only memory (EEPROM), or other erasable electronic memory which is known in the art. If storage device 112 comprises more than one storage device, various combinations of technologies could be used to store information relating to the operation of computing device 100.

In one embodiment, storage device 112 is divided into two or more storage areas, denoted in FIG. 1 as application storage area 106, programmable storage area 108, and permanent storage area 110. Application storage area 106 stores software programs, generally in executable form, and associated data.

Programmable storage area 108 stores information on a temporary basis. For example, data generated by software from application storage area 106 may generate data to be used by a computing device operator and could be stored in programmable storage area. Programmable storage area could also be used to  
5 store available software temporarily until various authentication procedures have been accomplished. The available software stored in programmable storage area 108 may then be erased or moved to application storage area 106 depending on the results of the authentication processes. Permanent storage area 110 comprises an area of storage device 112 where the kernal is stored. This area of  
10 storage device 112 is generally not able to be altered by processor 102, unlike application storage area 106 and programmable storage area 108. In addition, permanent storage area 110 stores an "authentication flag", which is explained below, in one embodiment.

After receiving instructions to load the available software, either from  
15 interface 104 or stored in programmable storage area 108, processor 102 determines whether or not resident software corresponding to the available software has been authenticated, in one embodiment. In another embodiment, processor 102 determines if resident software unrelated to the available software has been authenticated. For example, processor 102 would determine if an  
20 operating system software has been authenticated prior to loading an application such as an encryption program or a word processing program. Resident software is defined herein as software that is stored in storage device 112 and capable of being executed by processor 102.

Authentication is a well-known technique for verifying that software from  
25 a "trusted source" has not been altered. Authenticating software involves appending an alpha-numeric authentication code to software in which a recipient's "private key", or private code, is used to generate the alpha-numeric authentication code. When the software is received by the recipient, in this case computing device 100, the software may be authenticated by computing device  
30 102 performing an authentication procedure on the authentication code. In one embodiment, the authentication procedure comprises running a cyclic redundancy check (CRC). In another embodiment, the authentication procedure comprises running a secure hashing algorithm (SHA). Both methods are well known in the art. Of course, other known methods may be used to authenticate  
35 software provided to computing device 100.

The authentication process may be aided by the presence of an "authentication flag". An authentication flag is an indication that at least one piece of authenticated software has been loaded into computing device 100. In one embodiment, the authentication flag comprises an indication in permanent

storage area 110. For example, as software is loaded onto computing device 100, it is first checked by processor 102 to determine if it is authenticated or not. If processor 102 determines that the software is authenticated, the authentication flag is set in permanent storage area 110 by processor 102. Once the authentication flag is set, it normally cannot be reset. Therefore, once an authenticated piece of software has been loaded onto computing device 100, any further software to be loaded onto computing device 100 will have to be authenticated. Otherwise it will be rejected, as explained below.

In another embodiment, the authentication flag comprises a hardware "fuse". A hardware fuse is a well known device which generally comprises a conductor capable of being destroyed by an electric current. In the present case, if processor 102 determines that a piece of authenticated software has been loaded onto computing device 100, processor 102 sends a signal to "set" the hardware fuse, i.e., destroy the conductor relating to the fuse. When processor 102 is presented with available software to be loaded onto computing device 100, the fuse is checked to determine if it has been previously blown, or set. If it is set, this indicates that at least one piece of authenticated software has been loaded onto computing device 100, and that only authenticated software can be loaded onto computing device 100. This method of determining the authentication status of computing device 102 is advantageous in that processor 102 does not have to perform an authentication procedure on resident software each time software is available to be loaded onto computing device 100.

In any case, the authentication flag may be set in a number of different ways. In one embodiment, the authentication flag is set only if pre-selected software is loaded into computing device 100 and the pre-selected software is authenticated. In another embodiment, the pre-selected software does not have to be authenticated in order for processor 102 to set the authentication flag. In another embodiment, the authentication flag may be set manually, usually by a factory technician. In this case, computing device 100 is manufactured for the purpose of only receiving authenticated software. The authentication flag may be set by a technician generally by connecting a computer to interface 104 and running a computer program which sets the authentication flag.

In one embodiment, processor 102 performs authentication on software corresponding to the available software. Corresponding software refers to a previously release version of the same software. For example, corresponding software to Microsoft Word 7.0 comprises Microsoft Word 6.0, Microsoft Word 5.0, as well as any versions previously released by Microsoft prior to version 7.0. If the resident software has not been authenticated, processor 102 loads the available software from either programmable storage area 108 or interface 104, as

the case may be, to application storage area 106. If the resident software is authenticated, then processor 102 determines whether or not the available software is authenticated. If the available software is also authenticated, processor 102 loads the available software into application storage area 106. If the available software is not authenticated, processor 102 rejects the available software and it is not loaded onto computing device 100.

In another embodiment, processor 102 performs authentication on software unrelated to the available software. In this embodiment, the unrelated software generally controls major functions of computing device 100. For example, the unrelated software comprises operating system software stored in application storage device 106. The operating system software could be checked for authentication prior to any available software being loaded onto computing device 100. If the unrelated software has not been authenticated, processor 102 proceeds to load the available software into application storage area 106. This embodiment may be used if a previous version of the available software is not stored in application storage device 106, or it may be used irregardless of whether or not a previous version of the available software has been already loaded onto computing device 100.

If the unrelated software is not authenticated, processor 102 loads the available software into application storage area 106. If the unrelated software is authenticated, then processor 102 determines whether or not the available software is authenticated. If the available software is not authenticated, processor 102 rejects the available software, and it is not loaded into computing device 100. If the available software is authenticated, then processor 102 loads the available software into application storage area 106.

In yet another embodiment, processor 102 checks for a previous version of the available software and, in addition, checks software unrelated to the available software. In this embodiment, authentication is performed on both the previous version of the available software if it is already loaded onto computing device 100 and on the unrelated software. Processor 102 will then determine whether or not to load the available software based on the results of the authentications. For example, if both software programs are not authenticated, processor 102 will generally load the available software into application storage area 106. If both software programs are authenticated, then processor 102 generally will reject the available software and it will not be loaded into computing device 100. If either software programs are authenticated, then processor 102 may or may not load the available software into computing device 100, based upon a pre-defined methodology. For example, if the operating system software is authenticated, but the related software is not authenticated, then the available software will not be

loaded unless it is also authenticated. In this embodiment, a separate authentication flag could be used to indicate the authentication status of each software program.

FIG. 2 is a flow diagram illustrating one embodiment of the method for configuration management for a computing device. In step 200, software to be loaded onto computing device 100 is provided to computing device 100 through interface 104. This software is herein referred to as available software.

In step 202, processor 102 determines whether or not resident software stored in application storage device 106 has been authenticated, using one or more techniques described above. If the resident software is not authenticated, processor then determines whether or not the available software is authenticated, as shown in step 204. Again, processor 102 performs authentication on the available software as described above. If the available software is authenticated, an authentication flag is set in step 206. The authentication flag is an indication that computing device 100 may only accept authenticated software. After the authentication flag has been set, the available software is loaded onto computing device 100, as shown in step 208. If the available software is authenticated, it is loaded onto computing device 100 without setting the authentication flag, as shown in step 210.

If the resident software has been authenticated by processor 102 in step 202, the available software is checked for authentication in step 212. If the available software is not authenticated, processor 102 rejects the available software, as shown in step 214, and the available software is not loaded onto processing device 100. If the available software is authenticated, then processor 102 loads the available software onto computing device 100, as shown in step 216.

The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

**WE CLAIM:**